1 Introduction

The purpose of this assignment is for you to become familiar with Scheme, DrScheme, writing recursive functions, and the concept of a meta-circular interpreter.

Before starting this assignment, set your DrScheme language level to Standard (R5RS).

All your function definitions should be pure, i.e. they should not use any of Scheme’s imperative features such as set!. Also, never use iteration, always recursion.

Every function should be commented. At the very least, the comments should state what the function does, which arguments it takes, and what result it produces.

You may work in teams of two.

This assignment is graded out of 100. It is worth 10% of your final grade.

2 Simple functions

1. Define a recursive function \texttt{(copy-string s n)} which returns a string consisting of \texttt{n} copies of the string \texttt{s}: \[10 \text{ points}\]

   \texttt{(define (copy-string s n)}
   \texttt{(cond}
   \texttt{[(<= n 0) ...]}
   \texttt{[(= n 1) ...]}
   \texttt{[else ...]}
   \texttt{})
   \texttt{)}

   Your function should have the following behavior:

   \[
   \begin{align*}
   > & (\text{copy-string "hello" -1}) \\
   & "" \\
   > & (\text{copy-string "hello" 0}) \\
   & "" \\
   > & (\text{copy-string "hello" 1}) \\
   & "hello" \\
   > & (\text{copy-string "hello" 2}) \\
   & "hellohello"
   \end{align*}
   \]
2. Define a recursive function \texttt{(power-of-two? n)} which returns \texttt{#t} if \texttt{n} is a power of two (i.e. $n = 2^m$), and \texttt{#f} otherwise: \[10\text{ points}\]

\begin{verbatim}
(define (power-of-two? n)
   (cond
       ....
   ))
\end{verbatim}

Your function should have the following behavior:

\begin{itemize}
   \item \texttt{> (power-of-two? 0)} \hspace{1cm} \texttt{#f}
   \item \texttt{> (power-of-two? -4)} \hspace{1cm} \texttt{#f}
   \item \texttt{> (power-of-two? 1)} \hspace{1cm} \texttt{#f}
   \item \texttt{> (power-of-two? 2)} \hspace{1cm} \texttt{#t}
   \item \texttt{> (power-of-two? 3)} \hspace{1cm} \texttt{#f}
   \item \texttt{> (power-of-two? 4)} \hspace{1cm} \texttt{#t}
   \item \texttt{> (power-of-two? 6)} \hspace{1cm} \texttt{#f}
   \item \texttt{> (power-of-two? 8)} \hspace{1cm} \texttt{#t}
\end{itemize}

3 A Metacircular Interpreter

Extend the metacircular interpreter from lecture notes #40 with the functionality below. In all cases you can assume that the input programs are correct, i.e. you don’t need to check for error conditions.

\begin{itemize}
   \item \texttt{(display arg)}: \hspace{1cm} \[10\text{ points}\]
   \item \texttt{: (display 55))}
   \item \texttt{5555}
   \item \texttt{Note that display returns the value it has just printed, so the output should be 5555!}
   \item \texttt{(newline)}: \hspace{1cm} \[10\text{ points}\]
   \item \texttt{: (newline))}
   \item \texttt{()}
   \item \texttt{newline returns null.}
\end{itemize}
3. (begin arg1 ... argn):

   > (mEval '(begin (display 55) (newline) (display 66) (newline) (+ 4 5)))
   55
   66
   9

   Note that the last value (9) is in the output not because display printed it, but because begin returns the last value evaluated.

4. (cond (expr1 arg1) ... (exprn argn)):

   > (mEval '(cond ((eq? 1 2) (display 55)) ((eq? 2 2) (display 66))))
   66

   Don't use Scheme's built-in cond-function in your implementation (you can use if, however)! Also add the constants #t and #f to the interpreter, so that you can say things like:

   > (mEval '(cond ((eq? 1 2) 44)
                   ((equal? (quote (1 2)) (quote (1 (2)))) 55)
                   (#t 66))
   66

5. (equal? expr1 expr2):

   > (mEval '(equal? (quote (1 (2))) (quote (1 (2)))))
   #t
   > (mEval '(equal? (quote (1 (2))) (quote (1 (2 (3)))))
   #f

   Don't use Scheme's built-in equal?-function in your implementation (you can use eq?, however)! In other words, you need to write a recursive version of equal? that implements deep equivalence.

4. Extension

Add define and variable references:

   > (mEval '(begin
               (define x 44)
               (display x) (newline)
               (define x (+ x 11))
               (display x) (newline)))
   44
   55

   To implement this functionality you need to add an argument Env to each function. Env stores current variable values. You can implement Env as an association-list of variable/value pairs.
5 Submission and Assessment

The deadline for this assignment is noon, Wed May 7. It is worth 10% of your final grade.

You should submit the assignment electronically using the Unix command

```shell
turnin cs520.5 interp.scm README.
```

| Don’t show your code to anyone, don’t read anyone else’s code, don’t discuss the details of your code with anyone. If you need help with the assignment see the instructor or the TA. |